望获 embedded OS

望 获 操 作 系 统

—— 高可靠嵌入式实时操作系统 ——

开发手册

（v1.0）

北京国科环宇科技股份有限公司

2024 年 8 月 16 日

前言

概述

本文档详细介绍了基于望获操作系统的开发、适配流程，使得用户将望获操作系统运行到自定义的板卡并进行二次开发。

读者对象

本文档主要适用于以下人员：

- 使用望获操作系统的用户
- 基于望获操作系统进行开发的开发者
- 其他需要了解望获操作系统的人员

文档历史记录

版本号	实施日期	编写人	修订摘要
1.0	2023.8.16	蔡纪良 王志强	初始版本

目　　录

1 系统概述

1.1 支持平台一览

- Phytium(2000，e2000，d2000，d2000v)
- x86
- Rockchip(3588 3568)

1.2 平台 BIOS 一览

表 1- 1 平台 BIOS 一览表

平台 \ BIOS	grub	uboot
Phytium	Yes	Yes
Rockchip	No	Yes
x86	Yes	No

1.3 启动说明

几乎所有平台都支持从启动盘启动，进行试用或安装（将系统安装到板卡的其他存储介质，如 NVME、EMMC 等）。

Rockchip3588 平台对于启动盘的支持，比较特殊。该平台若想从启动盘启动，必须提前将 Uboot 镜像烧录到板卡 EMMC 的第一分区，那么，在烧录 Uboot 镜像时，将整个系统烧录到 EMMC 是更加便捷的。故该平台在使用时，通常首次烧录为整个系统镜像的烧录（包括 Uboot 镜像），后续在保证 Uboot 镜像正常运行的情况下，可以通过启动盘启动升级系统。

1.4 镜像格式

望获操作系统的常用镜像格式为 wic，其通常包含两个分区（个别平台，略有不同），第一个分区为启动分区，内含诸如 Linux 内核、设备树、initramfs 等文件；第二分区为根文件系统。

Rockchip3588 平台的系统镜像，有 update.img 和 wic 格式两种，不同点在于，只有 update.img 可以用于无系统板卡的首次烧录，wic 常用于制作启动盘进行升

级系统。wic 制做的启动盘，要求板卡中的 BIOS（在 Rockchip 中为 Uboot）运行正常；而 update.img 则无此限制。

1.5 系统烧录

参见《望获操作系统使用说明手册》烧录系统到板卡。

2 启动调试

2.1 分区说明

2.1.1 启动分区

启动分区通常为第一分区（Rockchip 平台除外），包含：

1. 内核镜像（Image 或 zImage 等，各平台略有不同）
2. Initrd 镜像（cpio.gz 或 cpio.gz.u-boot 格式）
3. 设备树（.dtb 文件）
4. 引导程序及配置文件等

2.1.2 根文件系统分区

根文件系统分区通常为第二分区（Rockchip 平台除外），包含 Systemd、C 库、shell 以及各种系统服务、应用软件。

2.1.3 Rockchip 特别说明

Rockchip 平台，第一分区为 Uboot 镜像（二进制文件，不可挂载）；第二分区为启动分区，第三分区为根文件系统分区。

2.2 文件修改替换

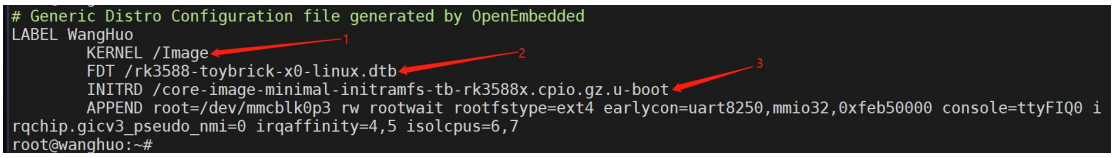
本章介绍对望获系统内的文件进行修改或替换的方式，用于支持用户自定义系统的需求。文件的修改和替换，与硬件平台的 BIOS 类型（U-boot 或 Grub）和镜像格式有关，相关信息参见 1.4　镜像格式和 1.2　平台 BIOS 一览章节。

2.2.1 启动文件定位

对望获系统内的文件进行修改或替换的基础，是要定位到系统的启动，使用了哪些文件，存放在哪些分区，本小节主要讲解如何定位这些文件的列表和位置。

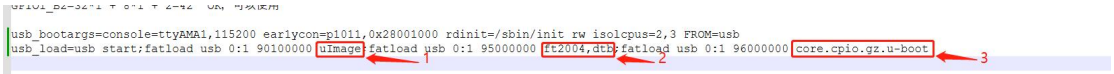
启动文件的定位，和 BIOS 类型有关系。如果 BIOS 类型是 U-boot，对于

Rockchip 平台来说，需要查看文件 extlinux.conf 的内容，里面详述了启动文件的名字和位置，该文件位于第二分区的 extlinux 文件夹里，其内容如下图所示。



数字 1 就是内核文件，名字是 Image，数字 2 就是设备树文件，名字是 rk3588-toybrick-x0-linux.dtb ； 数 字 3 就 是 initramfs ， 名 字 是 core-image-minimal-initramfs-tb-rk3588x.cpio.gz.u-boot；这 3 个文件的位置都是位于启动分区（第二分区）根目录下。

对于 Phytium 平台，没有启动文件，需要看 U-boot 环境变量，如下图所示。



我们以从 U 盘启动盘启动系统为例，数字 1 是内核文件，名字是 uImage；数字 2 是设备树文件，名字是 ft2004.dtb；数字 3 是根文件系统，名字是 core.cpio.gz.u-boot。这 3 个启动文件的位置都是位于 U 盘启动盘第一分区根目录下。

如果 BIOS 类型是 Grub，对于 x86 平台和 Phytium 平台，需要查看 grub.cfg 文件内容，里面详述了启动文件的名字和位置，该文件位于第一分区（启动分区）EFI/BOOT/文件夹里，其内容如下图所示。

数字 1 和 3 就是内核文件，名字是 bzImage;数字 2 和 4 就是根文件系统，名字是 core-image-minimal-initramfs-intel-corei7-64.cpio.gz;只不过 1 和 2 对应直接从安装盘，例如 U 盘启动系统的情况，而 3 和 4 对应通过安装盘，将系统安装到板卡存储介质上，然后从该介质启动系统的情况。bzImage 和 core-image-minimal-initramfs-intel-corei7-64.cpio.gz 的位置都是位于启动分区（第一分区）根目录下。

对于各个平台的 BIOS 支持情况，参见 1.2　平台 BIOS 一览。

2.2.2 镜像直接修改替换

本小节主要介绍直接修改系统镜像，适用于镜像格式为wic的操作系统镜像。我们以所在操作系统ubuntu，所用镜像所在平台是Rockchip为例。

我们使用kpartx和mount工具，请确保系统安装了这2个工具。将对应的wic镜像文件拷贝到工作目录。我当前所用的wic镜像的名字是wanghuo-image-standard-tb-rk3588x.wic。接下来按照下面数字编号的步骤进行。

1.使用kpartx工具创建设备映射。

```
sudo kpartx -av wanghuo-image-standard-tb-rk3588x.wic
```

运行此命令后，`kpartx`会输出类似如下的信息：

```
add map loop18p1 (253:0): 0 8192 linear 7:18 16384
add map loop18p2 (253:1): 0 6215680 linear 7:18 32768
add map loop18p3 (253:2): 0 9306754 linear 7:18 6258688
```

这些输出表示wic镜像中的分区已经映射到了设备文件，例如/dev/mapper/loop18p1和/dev/mapper/loop18p2以及/dev/mapper/loop18p3。

2.挂载对应分区。

```
sudo mount /dev/mapper/loop018p2 /mnt/boot
sudo mount /dev/mapper/loop018p3 /mnt/rootfs
```

请确保 /mnt/boot和/mnt/rootfs已经存在，否则创建它们。

3.访问挂载点。

通过ls /mnt/boot命令可以查看启动分区的内容，通过ls /mnt/rootfs可以查看根文件系统分区的内容。两个分区内的内容都可以进行替换修改，结束后请执行继续以下4，5两步。

4.卸载对应分区。

```
sudo umount /mnt/boot
sudo umount /mnt/rootfs
```

5.使用kpartx工具删除设备映射。

```
sudo kpartx -dv wanghuo-image-standard-tb-rk3588x.wic
```

2.2.3 启动介质内修改替换

本小节主要介绍在启动介质中修改替换文件的方法。适用于启动介质可以插拔的情况，如（包括但不限于）：

- NVME盘

- SD 卡
- U 盘

以 U 盘为例，将 U 盘插入 Ubuntu 系统中。通过执行 lsblk 命令会看到系统当中出现了 sda 设备（sda 名字不唯一，也有可能是别的名字，例如 sdb 等）。若该盘内已经烧写的望获操作系统，则应该会存在如 2.1　分区说明章节所描述的分区。本章节以 Rockchip 为例，故该设备有 3 个分区，分别是/dev/sda1，/dev/sda2，/dev/sda3。

1. 挂载对应分区。

sudo mount /dev/sda2 /mnt/boot

sudo mount /dev/sda3 /mnt/rootfs

请确保 /mnt/boot 和/mnt/rootfs 已经存在，否则创建它们。

2.访问挂载点。

通过 ls /mnt/boot 命令可以查看启动分区的内容，通过 ls /mnt/rootfs 可以查看根文件系统分区的内容。两个分区内的内容都可以进行替换修改。

3.卸载对应分区。

sudo umount /mnt/boot

sudo umount /mnt/rootfs

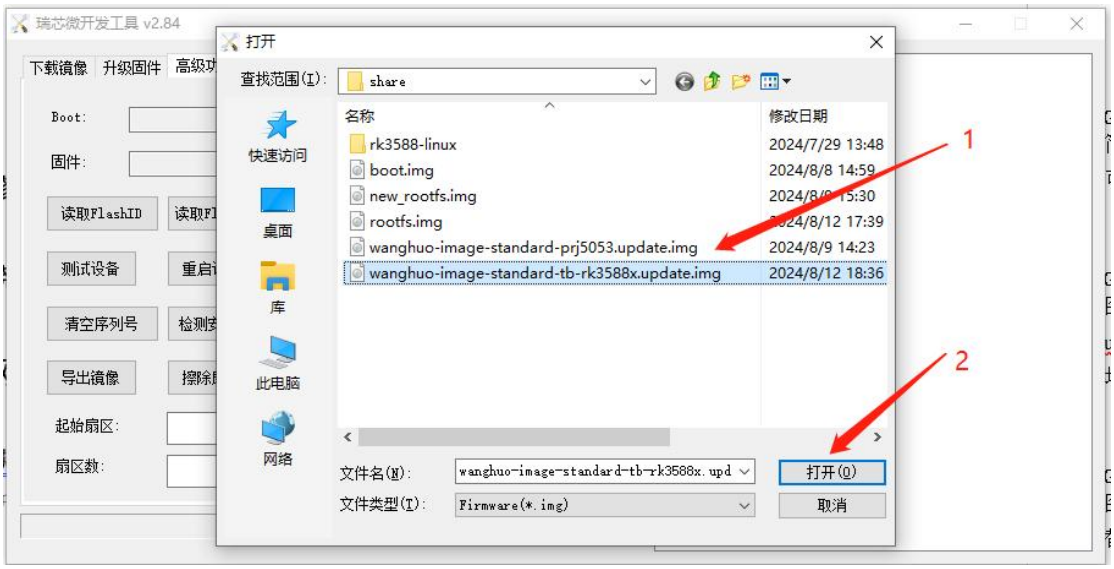
2.2.4　Rockchip 修改替换

由于 Rockchip 平台镜像的特殊性（update.img），需要先解包，才能对文件进行修改和替换。本小节主要介绍 Rockchip 镜像的解包方式以及解包后的烧录方法。

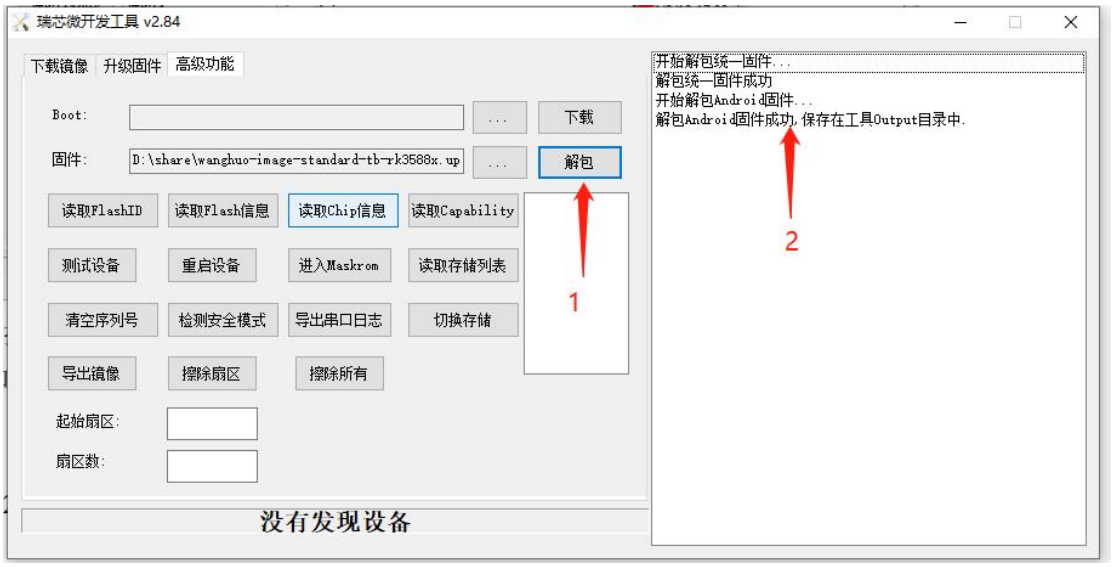
2.2.4.1　解包

1.我们打开瑞芯微官方给的解包工具 RKDevTool.exe，先点击高级功能，然后点击 解包 功能左边的三个小黑点，以此来选择要解包的.img 文件，如下图所示。

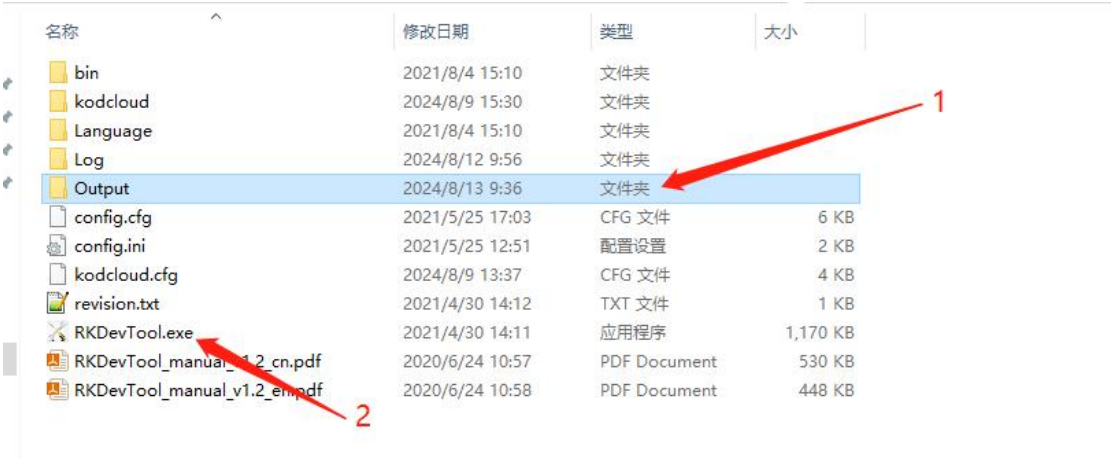
2.选择本地文件 wanghuo-image-standard-tb-rk3588x.update.img，然后点击打开，如下图所示。



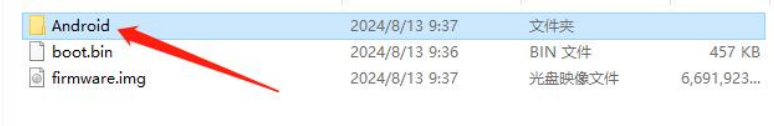
3.打开之后，点击 解包 ，等待解包结束，右侧的空白栏会显示实时的解包进度，如下图所示，数字 1 代表点击 解包 功能，数字 2 显示实时进度，此处显示解包已经结束，解包后的文件在 Output 目录中。



4.打开与瑞芯微官方给的解包工具 RKDevTool.exe 相同目录的 Output 文件夹，，如下图所示，数字 1 就是我们要选择的 Output 文件夹，数字 2 就是我们用的解包工具 RKDevTool.exe，两者在相同目录。



5.进入 Output 文件夹后，选择该文件夹下的 Android 文件夹，Android 文件夹下的文件就是我们解包后的各个文件，进入 Output 文件夹下的 Android 文件夹，如下图所示。



进入 Android 文件夹，其内容如下图所示，其中经常用到的就是 boot.img 和 rootfs.img 文件，boot.img 包含启动分区里的内容，rootfs.img 包含根文件系统分区里的内容。

2.2.4.2 修改替换

我们要修改解包后的 boot.img 和 rootfs.img 文件，需要先挂载这 2 个文件，然后再修改替换里面的内容，修改完后再卸载就行了。

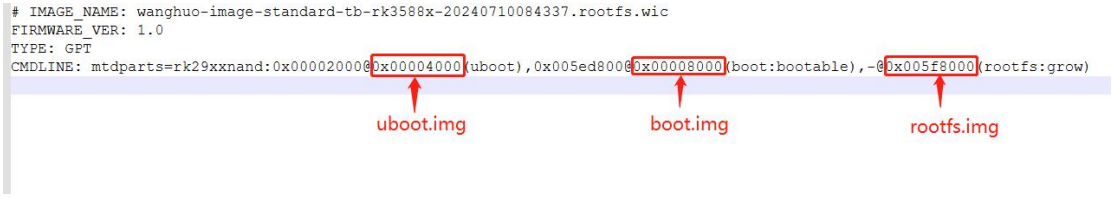
2.2.4.3 烧录

从 2.2.4.1 解包这一节，我们获取到了解包后的各个文件，分别是 boot.img，loader.bin，package-file，parameter，rootfs.img，uboot.img。烧录的过程就是把这几个文件写入到可启动的存储设备各自所在正确的地址上（package-file 可以不用烧录）。

1.获取各个文件的烧录地址

loader.bin 和 parameter 文件的烧录地址一般是固定的 0 地址。

boot.img，rootfs.img，uboot.img 文件的烧录地址需要从 parameter 文件来查看，我们打开 parameter，建议用 Notepad++工具打开，可以看到如下图所示的内容。



红框里的就是各个文件的烧录地址，比如我这里可以看到 3 个文件的烧录地址分别是：

uboot.img ： 0x4000

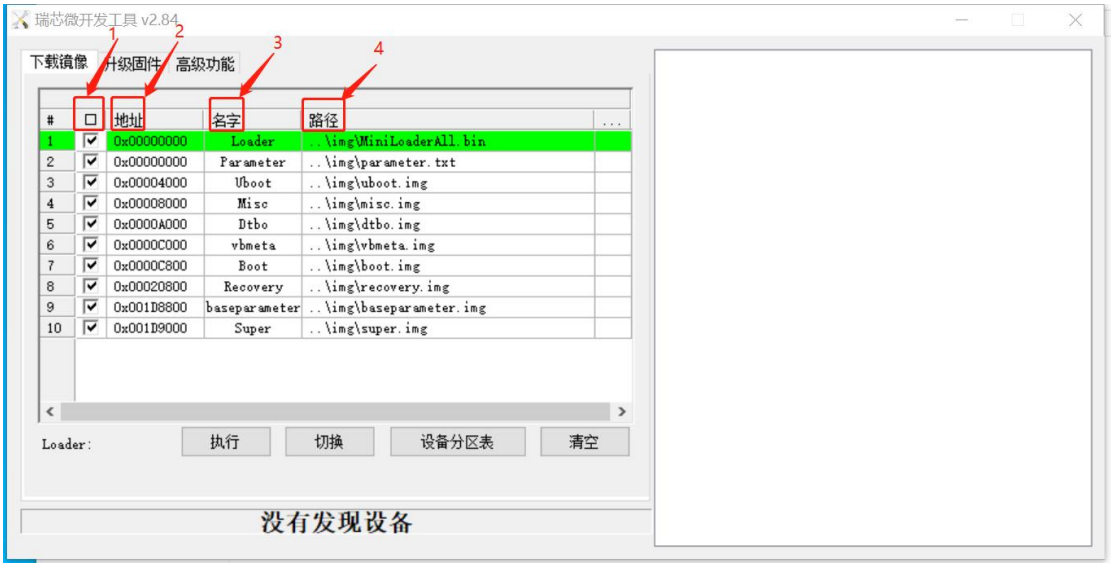
boot.img： 0x8000

rootfs.img： 0x5f8000

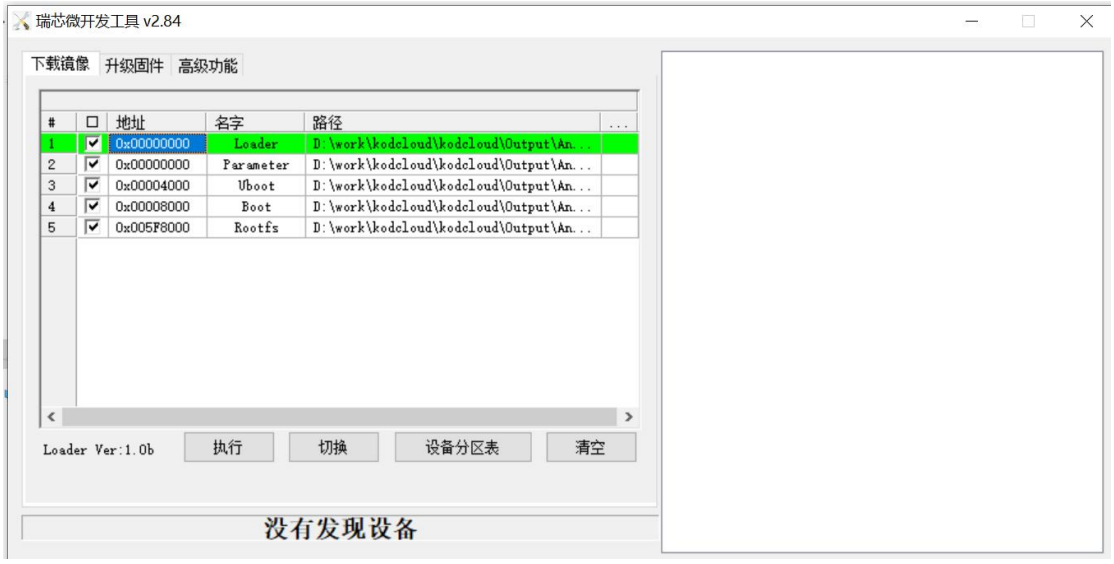
2.建立烧录镜像的配置

我们打开瑞芯微官方给的解包工具 RKDevTool.exe，点击左上角 下载镜像 ，会看到如下内容，每一行都代表一个可供烧录的文件及其属性。其中数字 1，也就是□这一列，打上✓，代表烧录的时候，会把该文件烧录进去，不打就不会烧录；数字 2 地址 代表我们要烧录的文件的地址，这个我们已经获取到了，

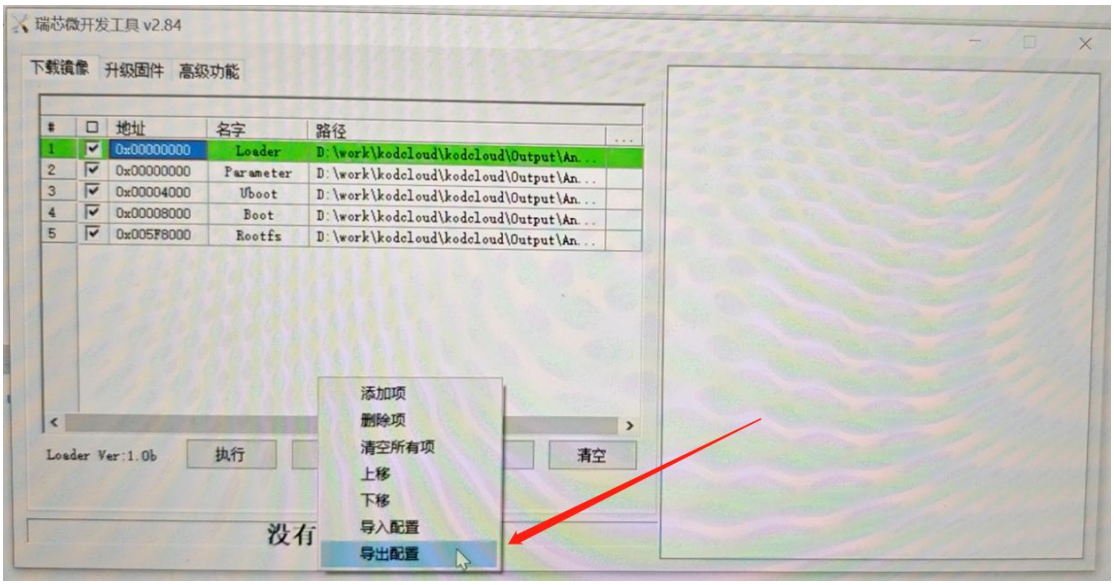
双击对应位置就能更改；数字 3 名字 就是我们要给烧录的文件定义一个名字，和原文件名字是否一样都可以，双击对应位置就可以更改；数字 4 路径 就是我们要烧录的文件所在的位置，双击这一列右边的那一列位置，也就是路径 右边 3 个黑点所对应的那一列，就能够选择我们的烧录文件。



比如更改完之后对应的下载镜像的配置可能如下图所示。注：具体情况需要按照上边讲述的逻辑，具体确定所烧录的地址。



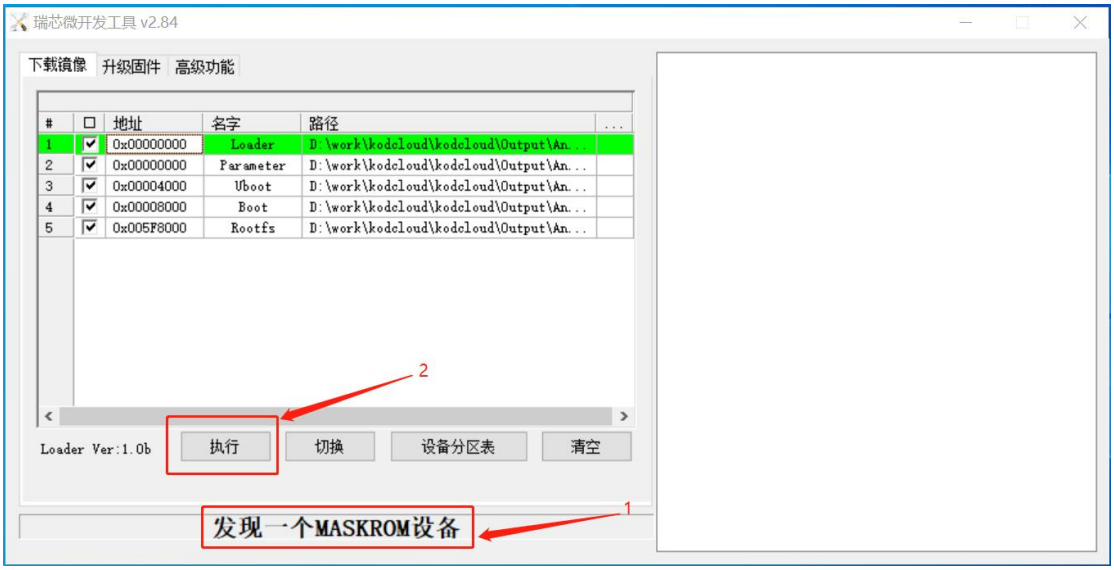
我们可以把我们的下载镜像的配置导出为一个文件保存一下，这样的话下次再烧录文件，打开 RKDevTool.exe 后，直接导入我们之前导出的配置文件就可以了，接下来我们导出配置文件：鼠标放到切换上边的空白位置，右键单击，左键选择导出配置，然后把配置文件导出到自己的一个合适的地方就可以，如下图所示。



等到下次打开 RKDevTool.exe 后，鼠标放到切换上边的空白位置，右键单击，左键选择导入配置，将上次我们导出的配置文件选择一下，就可以了。

3.烧录镜像

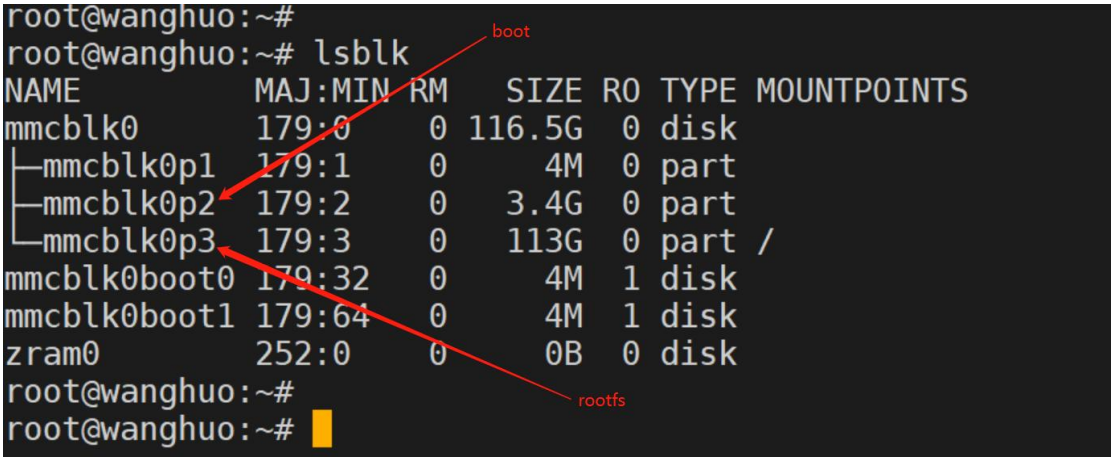
让设备进入 MASKROM 烧录模式，选择好我们要烧录的镜像，点击 执行 就可以了，如下图所示。



2.2.5 网络修改替换

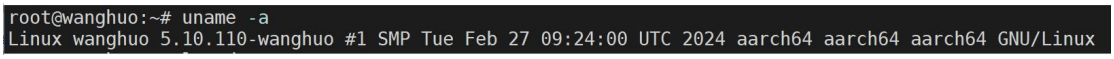
当望获系统可以成功启动且板卡网络工作正常时，可以使用网络 ssh 登录到板卡，将所替换文件传输到板卡，挂载板卡上的磁盘设备，替换里边的文件。具体步骤如下所示。

1.通过 ssh 登录板卡，查看磁盘设备，如下图所示。



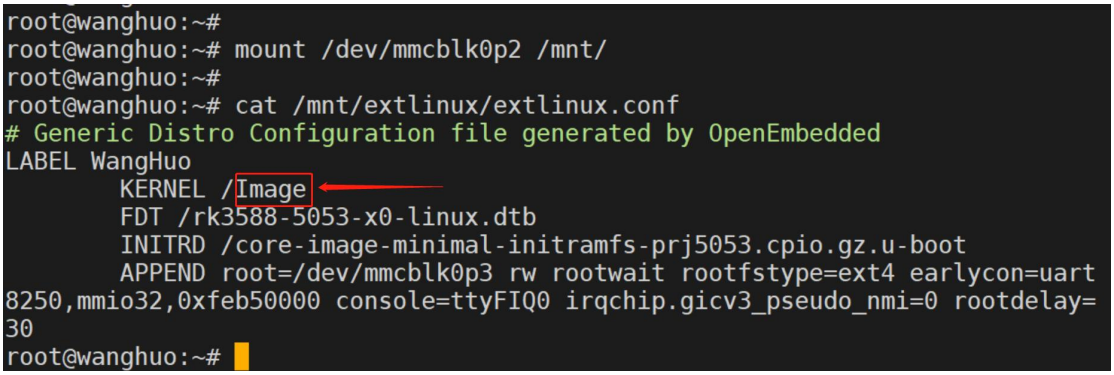
其中/dev/mmcblk0p2 是启动分区，也就是标识当中的 boot，/dev/mmcblk0p3 是根文件系统分区，也就是标识当中的 rootfs。接下来我们要替换 boot 分区里的内核文件。

为了确定我们替换的内核文件是有效的，我们先通过 uname -a 查看一下原本的内核版本是多少，如下图所示，可以看到当前内核版本是 5.10.110,接下来我们更换一下内核版本为 5.10.160 的。



2.确定要替换的内核文件名字

将 boot 分区挂载，查看启动文件内容，确定内核文件的名字和位置，具体的启动文件名字请参阅 2.2.1　启动文件定位，实际操作如下图所示。



我们能够看到所用的内核文件名字是 Image，其位置位于启动分区顶层目录下，其实也就是我们现在的 /mnt 目录下，该目录下还有其它文件，如下图所示。

```
root@wanghuo:~#
root@wanghuo:~# ls /mnt/
backup.conf
core-image-minimal-initramfs-prj5053-20240809060221.cpio.gz
core-image-minimal-initramfs-prj5053-20240809060221.cpio.gz.u-boot
core-image-minimal-initramfs-prj5053.cpio.gz
core-image-minimal-initramfs-prj5053.cpio.gz.u-boot
extlinux
Image
Image-5.10.110-wanghuo
Module.symvers-5.10.110-wanghuo
rk3588-5053-x0-linux.dtb
System.map-5.10.110-wanghuo
root@wanghuo:~#
root@wanghuo:~#
```

红色箭头就是我们要替换的内核文件 Image。

3.通过 scp 命令把我们的内核文件拷贝到这个位置直接覆盖，或者拷贝到板卡的别的位置，再 cp 过来覆盖，方式不唯一，只要能达到目的就可以。然后重启系统。

4.系统重启后，通过 ssh 连接后台，执行 uanme -a 命令，可以看到是我们替换的 5.10.160 版本的内核文件，说明我们的替换是成功的。

```
root@wanghuo:~# uname -a
Linux wanghuo 5.10.160-wanghuo #2 SMP Tue Aug 13 14:16:25 CST 2024 aarch64 a
arch64 aarch64 GNU/Linux
root@wanghuo:~#
```

2.3 设备树调试

前面两个章节，描述了文件的定位（位置和名称）、替换的方式，本章节重点描述的是，设备树文件内容本身的修改、编译等。

2.3.1 源码获取

对于 Rockchip 平台，源码地址如下：

https://github.com/rockchip-linux/kernel/tree/develop-5.10

对于 Phytium 平台，需要从硬件厂商获取。

2.3.2 编译替换

1.修改编译设备树。

获取到内核源码后，进入内核源码根目录下，板卡厂商的设备树文件一般在 arch/arm64/boot/dts/对应厂商，例如我们以 Rockchip 为例，其设备树文件位于 arch/arm64/boot/dts/rockchip 目录下。在这里我们以该目录下 rk3588-linux.dtsi 文件为例，打开该文件，里面有如下内容。

```
fiq_debugger: fiq-debugger {
        compatible = "rockchip,fiq-debugger";
        rockchip,serial-id = <2>;
        rockchip,wake-irq = <0>;
        /* If enable uart uses irq instead of fiq */
        rockchip,irq-mode-enable = <1>;
        rockchip,baudrate = <115200>;  /* Only 115200 and 1500000 */
        interrupts = <GIC_SPI 423 IRQ_TYPE_LEVEL_LOW>;
        pinctrl-names = "default";
        pinctrl-0 = <&uart2m0_xfer>;
        status = "okay";
};
```

可以看到 fiq_debugger 节点的状态（status）是 okay 的，如果我们不想使用这个节点，可以将其 disable 掉，具体做法就是给 status 属性赋值为”disabled”，修改后的内容如下：

```
fiq_debugger: fiq-debugger {
        compatible = "rockchip,fiq-debugger";
        rockchip,serial-id = <2>;
        rockchip,wake-irq = <0>;
        /* If enable uart uses irq instead of fiq */
        rockchip,irq-mode-enable = <1>;
        rockchip,baudrate = <115200>;  /* Only 115200 and 1500000 */
        interrupts = <GIC_SPI 423 IRQ_TYPE_LEVEL_LOW>;
        pinctrl-names = "default";
        pinctrl-0 = <&uart2m0_xfer>;
        status = "disabled";
};
```

修改完之后，编译内核设备树，生成 dtb 文件。

2.替换设备树文件。

替换设备树和替换内核文件的过程是一样的，首先确定启动文件的内容，然后根据启动文件确定需要替换的设备树文件名字和位置，最后将新的设备树文件拷贝到板卡覆盖原设备树文件，最后重启系统。

具体可参考 2.2　文件修改替换。

2.4 Initramfs 调试

initramfs（initial RAM filesystem）是一种用于引导 Linux 内核的初始文件系统。它在系统引导过程中作为临时根文件系统，主要用于在内核启动时加载所需的驱动程序和其他必要的资源，以便能够挂载最终的根文件系统。

2.4.1 禁用与使能

本章主要介绍 initramfs 的禁用与使能的方法。

1.对于使用 grub 的平台，需要修改 grub.cfg 文件，其位于启动分区 EFI/BOOT 下。如果是禁用 initramfs，只需要把 initrd 那一行用 # 字符注释掉就可以，如下图所示。

```
root@wanghuo:/mnt/EFI/BOOT# cat grub.cfg
# Automatically created by OE
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
default=WangHuo OS
timeout=10

menuentry 'WangHuo OS'{
linux /bzImage LABEL=WangHuo OS root=PARTUUID=84cf7a32-2299-497b-9c6d-31eedd
77a591   rootwait isolcpus=2 console=ttyS0,115200 console=tty0

#initrd /core-image-minimal-initramfs-intel-corei7-64.cpio.gz
}

root@wanghuo:/mnt/EFI/BOOT#
root@wanghuo:/mnt/EFI/BOOT#
```

如果是使能 initramfs，只需要把 initrd 那一行的 # 字符去掉就可以了，如下图所示。

```
root@wanghuo:/mnt/EFI/BOOT# cat grub.cfg
# Automatically created by OE
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
default=WangHuo OS
timeout=10

menuentry 'WangHuo OS'{
linux /bzImage LABEL=WangHuo OS root=PARTUUID=84cf7a32-2299-497b-9c6d-31eedd
77a591   rootwait isolcpus=2 console=ttyS0,115200 console=tty0

initrd /core-image-minimal-initramfs-intel-corei7-64.cpio.gz
}

root@wanghuo:/mnt/EFI/BOOT#
root@wanghuo:/mnt/EFI/BOOT#
```

2.对于使用 U-boot 的平台，如果是 Rockchip 的话，需要修改 extlinux.conf 文件内容，禁用 initramfs 的话，只需要把 INITRD 那一行用 # 字符注释掉就可以,如下图所示。

```
root@wanghuo:/mnt/extlinux#
root@wanghuo:/mnt/extlinux# cat extlinux.conf
# Generic Distro Configuration file generated by OpenEmbedded
LABEL WangHuo
        KERNEL /Image
        FDT /rk3588-5053-x0-linux.dtb
        #INITRD /core-image-minimal-initramfs-prj5053.cpio.gz.u-boot
        APPEND root=/dev/mmcblk0p3 rw rootwait rootfstype=ext4 earlycon=uart
8250,mmio32,0xfeb50000 console=ttyFIQ0 irqchip.gicv3_pseudo_nmi=0 rootdelay=
30
root@wanghuo:/mnt/extlinux#
root@wanghuo:/mnt/extlinux#
```

使能 initramfs 的话，只需要把 INITRD 那一行前面的 # 字符去掉就可以,如下图所示。

```
root@wanghuo:/mnt/extlinux#
root@wanghuo:/mnt/extlinux# cat extlinux.conf
# Generic Distro Configuration file generated by OpenEmbedded
LABEL WangHuo
        KERNEL /Image
        FDT /rk3588-5053-x0-linux.dtb
        INITRD /core-image-minimal-initramfs-prj5053.cpio.gz.u-boot
        APPEND root=/dev/mmcblk0p3 rw rootwait rootfstype=ext4 earlycon=uart
8250,mmio32,0xfeb50000 console=ttyFIQ0 irqchip.gicv3_pseudo_nmi=0 rootdelay=
30
root@wanghuo:/mnt/extlinux#
root@wanghuo:/mnt/extlinux#
```

如果是 Phytium 的话，如下图所示，截取了部分 U-boot 下的环境变量，其中只讨论通过 usb 启动系统的情况。

```
tftp_bootargs=console=ttyAMA1,115200 earlycon=pl011,0x28001000 rdinit=/sbin/init rw isolcpus=2,3 FROM=tftp
tftp_load=tftpboot 90100000 uImage;tftpboot 95000000 ft2004.dtb;tftpboot 96000000 core.cpio.gz.u-boot
usb_bootargs=console=ttyAMA1,115200 earlycon=pl011,0x28001000 rdinit=/sbin/init rw isolcpus=2,3 FROM=usb
usb_load=usb start;fatload usb 0:1 90100000 uImage;fatload usb 0:1 95000000 ft2004.dtb;fatload usb 0:1 96000000 core.cpio.gz.u-boot
usb_root_bootargs=console=ttyAMA1,115200 earlycon=pl011,0x28001000 init=/sbin/init root=/dev/sda2 rw rootwait FROM=usb_root
usb_root_load=usb start;fatload usb 0:1 90100000 uImage;fatload usb 0:1 95000000 ft2004.dtb
vendor=phytium

Environment size: 2379/4092 bytes
```

其中环境变量 usb_load，也就是箭头指向的 1，就是启用 initramfs 的情况，可以看到该变量最后是通过 fatload usb 0:1 96000000 core.cpio.gz.u-boot，来将 initramfs 读入到内存的固定位置的。该情况对应的内核启动参数请查看图中的环境变量 usb_bootargs；而环境变量 usb_root_load，也就是箭头指向的 2，就是禁用 initramfs 的情况，可以看到该变量并没有读入 initramfs。该情况对应的内核启动参数请查看图中的环境变量 usb_root_bootargs。最后我们通过 U-boot 启动系统的时候，选择 usb_load 或者 usb_root_load 就可以，当然也可以自己临时对一个变量进行修改。对比 usb_load 和 usb_root_load 两个环境变量内容的差异就是要修改的部分，请不要忘记内核启动参数也要同步修改。

2.4.2 修改 initramfs

如果 initramfs 是.cpio.gz.uboot 文件格式的，例如我们的文件是 core-image-minimal-initramfs-tb-rk3588x.cpio.gz.u-boot，按照如下方式解

压。

1.首先保存一下 core-image-minimal-initramfs-tb-rk3588x.cpio.gz.u-boot 文件的前 64 字节的 u-boot 信息,将其保存到 output_file 文件里。如下命令所示。

```
dd if=core-image-minimal-initramfs-tb-rk3588x.cpio.gz.u-boot of=output_file bs=1 count=64
```

执行完后，本目录会新增一个 output_file 文件

2. 去掉前 64 字节 uboot 后缀，执行下面的命令

```
dd if=core-image-minimal-initramfs-tb-rk3588x.cpio.gz.u-boot of=core-image-minimal-initramfs-tb-rk3588x.cpio.gz skip=64 bs=1
```

执行完后，本目录会新增一个 core-image-minimal-initramfs-tb-rk3588x.cpio.gz 文件

3.解压后缀为.gz 文件

```
gunzip core-image-minimal-initramfs-tb-rk3588x.cpio.gz
```

执行完后， core-image-minimal-initramfs-tb-rk3588x.cpio.gz 文件会变成 core-image-minimal-initramfs-tb-rk3588x.cpio 后缀的文件

4.执行如下命令从给定的归档文件 core-image-minimal-initramfs-tb-rk3588x.cpio 当中提取文件到指定的目录，这里我在当前目录新建了一个 tmp 目录，将其提取到当前的 tmp 目录当中。

```
sudo cpio -idm --directory=./tmp < core-image-minimal-initramfs-tb-rk3588x.cpio
```

5.修改文件并打包

进入 tmp 目录，修改好文件后，开始打包，将打包后的文件 new.cpio 保存到上一级目录，打包命令如下。

```
sudo bash -c 'find . -depth | cpio -o > ../new.cpio'
```

6.压缩文件。

进入要打包的文件 new.cpio 所在的目录，执行如下命令压缩文件。

```
gzip new.cpio
```

执行完后，new.cpio 会被压缩成 new.cpio.gz 文件

7. 添加 uboot64 字节的头信息，执行如下命令。

```
cat output_file new.cpio.gz > new.cpio.gz.u-boot
```

执行完后会生成最终我们修改后的 new.cpio.gz.u-boot 文件。

8.更改文件名字。

执行如下命令，把 new.cpio.gz.u-boot 改成原来的文件名字 core-image-minimal-initramfs-tb-rk3588x.cpio.gz.u-boot。

```
mv new.cpio.gz.u-boot core-image-minimal-initramfs-tb-rk3588x.cpio.gz.u-boot
```

如果 initramfs 是 .cpio.gz 文件，例如我们的文件是 core-image-minimal-initramfs-tb-rk3588x.cpio.gz，只需要参照上述 3，4，5，6，8 步骤

不过 8 步骤应该执行的命令是

mv new.cpio.gz core-image-minimal-initramfs-tb-rk3588x.cpio.gz

替换文件请参考 2.2 文件修改替换章节。

2.5 内核启动参数调试

内核启动参数的修改，和系统使用的启动文件有关，需要修改对应的启动文件内容，寻找对应的启动文件，具体请参考 2.2.1 启动文件定位。

1.系统使用 grub.cfg 启动文件的情况。

例如以 x86 平台为例，如下图所示，如果是通过启动盘启动的，则需要修改 1 所指向的内容，新加启动参数就在该行末尾增加就行；如果是把系统安装在板卡的存储介质上，则需要修改 2 所指向的内容，新加启动参数就在该行末尾增加就行。

```
# Automatically created by OE
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
default=WangHuo OS
timeout=10

menuentry 'WangHuo OS'{
linux /bzImage LABEL=WangHuo OS root=PARTUUID=c0508a01-723a-4c5f-96a3-e2f37d714846  rootwait console=ttyS0,115200 console=tty0

initrd /core-image-minimal-initramfs-intel-corei7-64.cpio.gz
}

menuentry 'install'{
linux /bzImage LABEL=install-efi root=PARTUUID=c0508a01-723a-4c5f-96a3-e2f37d714846  rootwait console=ttyS0,115200 console=tty0

initrd /core-image-minimal-initramfs-intel-corei7-64.cpio.gz
}
~
~
```

2.系统使用 extlinux.conf 启动文件的情况

如下图所示，需要修改箭头所指向的那一行内容，如果是新增启动参数，直接在该行末尾增加就行；如果是更改启动参数，则根据实际情况修改即可。

```
# Generic Distro Configuration file generated by OpenEmbedded
LABEL WangHuo
        KERNEL /Image
        FDT /rk3588-firefly-itx-3588j-mipi101-M101014-BE45-A1.dtb
        INITRD /core-image-minimal-initramfs-tb-rk3588x.cpio.gz.u-boot
        APPEND root=/dev/mmcblk0p3 rw rootwait rootfstype=ext4 earlycon=uart8250,mmio32,0xfeb50000 console=ttyFIQ0 irqchip.gicv3_pseudo_nmi=0
~
```

3.需要在 U-boot 阶段修改环境变量的情况。

如下图所示，箭头所指的是 U-boot 阶段通过 usb 启动系统时的内核启动参数，如果要新增加启动参数，直接在该行末尾增加就行。

```
tftp_bootargs=console=ttyAMA1,115200 earlycon=pl011,0x28001000 rdinit=/sbin/init rw isolcpus=2,3 FROM=tftp
tftp_load=tftpboot 90100000 uImage;tftpboot 95000000 ft2004.dtb;tftpboot 96000000 core.cpio.gz.u-boot
usb_bootargs=console=ttyAMA1,115200 earlycon=pl011,0x28001000 rdinit=/sbin/init rw isolcpus=2,3 FROM=usb
usb_load=usb start;fatload usb 0:1 90100000 uImage;fatload usb 0:1 95000000 ft2004.dtb;fatload usb 0:1 96000000 core.cpio.gz.u-boot
usb_root_bootargs=console=ttyAMA1,115200 earlycon=pl011,0x28001000 init=/sbin/init root=/dev/sda2 rw rootwait FROM=usb_root
usb_root_load=usb start;fatload usb 0:1 90100000 uImage;fatload usb 0:1 95000000 ft2004.dtb
vendor=phytium
```

3 模块开发

本章介绍在望获系统上如何开发一个简单的 hello 模块以及让该模块在系统启动后自动加载。

3.1 hello 模块

1.准备编译环境

进入望获系统 /usr/src/kernel 目录，准备编译环境，命令如下。

```
cd /usr/src/kernel
make prepare
make scripts
```

注意：前面的命令执行后，以后就不用再执行了，除非是系统本身发生了以下改变：

重新烧录了系统到板卡本身的存储介质。

重新安装了系统到板卡本身的存储介质。

重新制作了板卡的启动盘，而板卡是从启动盘启动的。

通过网络启动系统，而系统文件本身发生改变，比如替换等。

以上情况下需要重新准备编译环境。

2.编译模块代码

这里我们以简单的 hello 模块示例，现在我们把模块代码 hello.c 和对应的 Makefile 文件拷贝到系统的/home/root/module 目录，没有这个目录自己建一个。其中 hello.c 内容如下。

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/printk.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("wanghuo");
MODULE_DESCRIPTION("TEST module.");
MODULE_VERSION("1.00");
```

```
static int __init hello_init(void) {
   pr_info("hello world!\n");
   return 0;
}

static void __exit hello_exit(void) {
    pr_info("goodbye world!\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Makefile 内容如下

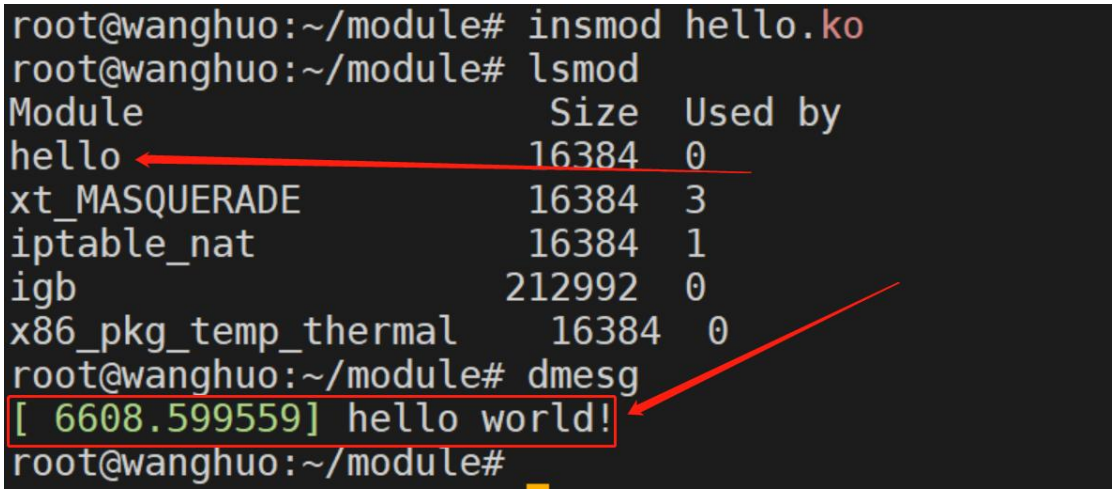
```
ifneq ($(KERNELRELEASE),)
      obj-m := hello.o
else
      SRC ?= /lib/modules/$(shell uname -r)/build
      PWD := $(shell pwd)
default:
      make -C $(SRC) M=$(PWD) modules
endif

clean:
      rm -rf .tmp_versions modules.order .tmp_versions .*.cmd *.o *.ko *.mod.c *.mod *.symvers
```

接下来我们编译模块代码，需要在/home/root/module 目录下执行 make 命令，结果如下图所示。

```
root@wanghuo:~/module# pwd
/home/root/module
root@wanghuo:~/module# ls
hello.c  Makefile
root@wanghuo:~/module# make
make -C /lib/modules/5.10.0-wanghuo/build M=/home/root/module modules
make[1]: Entering directory '/lib/modules/5.10.0-wanghuo/build'
  CC [M]  /home/root/module/hello.o
  MODPOST /home/root/module/Module.symvers
  CC [M]  /home/root/module/hello.mod.o
  LD [M]  /home/root/module/hello.ko
make[1]: Leaving directory '/lib/modules/5.10.0-wanghuo/build'
root@wanghuo:~/module#
```

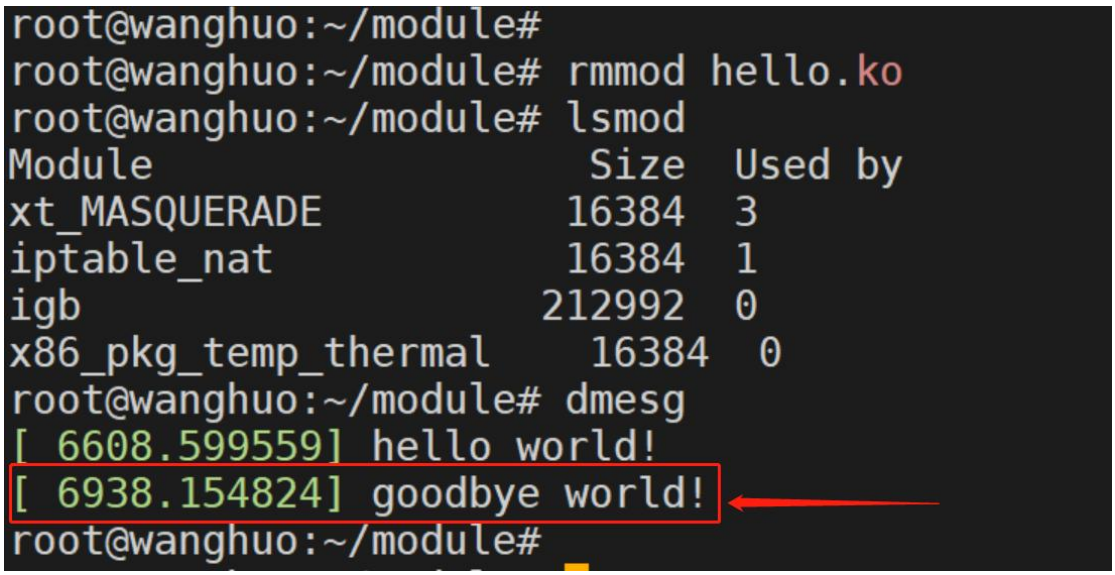
我们可以看到生成了 hello.ko 模块文件，然后我们加载该模块，执行 insmod

hello.ko，结果如下图所示。



通过执行 lsmod 命令，看到我们加载了 hello 模块；通过执行 dmesg 命令，看到打印出了 hello world!的信息。这说明我们编译的模块加载成功。

接下来我们通过执行 rmmod 命令卸载模块，如下图所示



通过执行 lsmod 命令，发现已经看不到 hello 模块了；通过 dmesg 命令，可以看到模块卸载的时候打印出来的信息 goodbye world!。这说明我们的模块已经卸载成功了。

3.2 模块自动加载

首先，将系统启动后需要自动加载的模块拷贝到/lib/modules/$(shell uname -r)目录下，$(shell uname -r)自己执行一下就可以看到具体的内容，例如我的就是5.10.110-wanghuo。建议把自己的模块拷贝到本目录下的 extra 目录，没有的话自己新建一个。这个是通用的树外模块的存放目录，例如我就把 hello.ko 拷贝到

/lib/modules/5.10.110-wanghuo/extra 目录，然后在上一级目录，也就是/lib/modules/5.10.110-wanghuo 目录下，执行 depmod -a 命令，重新生成一下内核所有模块的依赖关系。如下图所示

```
root@wanghuo:/lib/modules/5.10.110-wanghuo/extra# ls
hello.ko
root@wanghuo:/lib/modules/5.10.110-wanghuo/extra# cd ../
root@wanghuo:/lib/modules/5.10.110-wanghuo# ls
build   modules.alias      modules.builtin.alias.bin  modules.dep      modules.order    modules.symbols.bin
extra   modules.alias.bin  modules.builtin.bin        modules.dep.bin  modules.softdep  source
kernel  modules.builtin    modules.builtin.modinfo    modules.devname  modules.symbols
root@wanghuo:/lib/modules/5.10.110-wanghuo# depmod -a
root@wanghuo:/lib/modules/5.10.110-wanghuo#
```

然后进入系统 /etc/modules-load.d 目录，新建 hello.conf 文件，注意：务必使该文件名字.conf 的前缀和要加载的模块名字一样。例如，如果是 abcde.ko 模块，则需要新建 abcde.conf 文件。

在 hello.conf 文件里增加内容 hello。可以采用 echo hello > hello.conf 方式，也可以采用 vim 打开文件进行编辑的方式。总之，文件里面的内容必须是要加载的模块文件名字，例如，如果是 abcde.ko 模块。则需要在 abcde.conf 文件里新增内容 abcde。如下图所示。

```
root@wanghuo:/etc/modules-load.d#
root@wanghuo:/etc/modules-load.d# ls
fuse.conf  hello.conf  openvswitch.conf
root@wanghuo:/etc/modules-load.d#
root@wanghuo:/etc/modules-load.d# echo hello > hello.conf
root@wanghuo:/etc/modules-load.d#
root@wanghuo:/etc/modules-load.d# cat hello.conf
hello
root@wanghuo:/etc/modules-load.d#
root@wanghuo:/etc/modules-load.d# reboot
```

最后重启系统，就能够自动加载我们指定的模块了，如下图所示。

```
root@wanghuo:~#
root@wanghuo:~# lsmod
Module                  Size  Used by
bcmdhd               1613824  0
dhd_static_buf         16384  1 bcmdhd
openvswitch           135168  0
nsh                    16384  1 openvswitch
nf_conncount           36864  1 openvswitch
hello                  16384  0
root@wanghuo:~#
root@wanghuo:~# dmesg | grep hello
[    4.180397] hello: loading out-of-tree module taints kernel.
[    4.180866] hello world!
root@wanghuo:~#
```

这是系统重启后看到的结果，通过第一个箭头能够看到加载了 hello 模块；通过第二个箭头看到打印了模块里面的信息；这说明我们在系统启动的时候自动加载我们的模块已经成功了。